Optimal Zielonka-Type Construction of Deterministic Asynchronous Automata

Blaise Genest[1,2], Hugo Gimbert[3], Anca Muscholl[3], Igor Walukiewicz[3]

[1] CNRS, IPAL UMI, joint with I2R-A*STAR-NUS, Singapore
[2] CNRS, IRISA UMR, joint with Université Rennes I, France
[3] LaBRI, CNRS/Université Bordeaux, France

Abstract. Asynchronous automata are parallel compositions of finite-state processes synchronizing over shared variables. A deep theorem due to Zielonka says that every regular trace language can be represented by a deterministic asynchronous automaton. In this paper we improve the construction, in that the size of the obtained asynchronous automaton is polynomial in the size of a given DFA and simply exponential in the number of processes. We show that our construction is optimal within the class of automata produced by Zielonka-type constructions. In particular, we provide the first non trivial lower bound on the size of asynchronous automata.

1 Introduction

Zielonka's asynchronous automata [15] is probably one of the simplest, and yet rich, models of distributed computation. This model has a solid theoretical foundation based on the theory of Mazurkiewicz traces [9,4]. The key property of asynchronous automata, known as Zielonka's theorem, is that every regular trace language can be represented by a deterministic asynchronous automaton [15]. This result is one of the central results on distributed systems and has been applied in many contexts. Its complex proof has been revisited on numerous occasions (see e.g. [2,3,12,13,6] for a selection of such papers). In particular some significant complexity gains have been achieved since the original construction. This paper provides yet another such improvement, and moreover it shows that the presented construction is in some sense optimal.

The asynchronous automata model is basically a parallel composition of finite-state processes synchronizing over shared (state) variables. Zielonka's theorem has many interpretations, here we would like to consider it as a result about distributed synthesis: it gives a method to construct a deterministic asynchronous automaton from a given sequential one and a distribution of the actions over the set of processes. We remark that in this context it is essential that the construction gives a deterministic asynchronous automaton: for a controller it is the behaviour and not language acceptance that is important. The result has applications beyond the asynchronous automata model, for example it can be used to synthesize communicating automata with bounded communication channels

[11,7] or existentially-bounded channels [5]. Despite these achievements, from the point of view of applications, the biggest problem of constructions of asynchronous automata is considered to be their high complexity. The best constructions give either automata of size doubly exponential in the number of processes, or exponential in the size of the sequential automaton.

This paper proposes an improved construction of deterministic asynchronous automata. It offers the first algorithm that gives an automaton of size *polynomial* in the size of the sequential automaton and exponential only in the number of processes. We show that this is optimal for Zielonka-type constructions, namely constructions where each component has complete information about its history. For this we introduce the notion of *locally rejecting* asynchronous automaton and remark that all Zielonka-type constructions produce this kind of automata. To be locally rejecting means that a process should reject as soon as its history tells him that there is no accepting extension. We believe that a locally rejecting behavior is quite desirable for applications, such as monitoring or control. We show that when transforming a deterministic word automaton to a deterministic locally rejecting automaton, the exponential blow-up in the number of components is unavoidable. Thus, to improve our construction one would need to produce automata that are not locally rejecting.

For the upper bound we start from a *deterministic* (I-diamond) word automaton. We think that this is the best point of departure for a study of the complexity of constructing asynchronous automata: considering non deterministic automata would introduce costs related to determinization. The size of the deterministic asynchronous automaton obtained is measured as the sum of the sizes of the local states sets. It means that we do not take global accepting states into account. This is reasonable in our opinion, as it is hardly practical to list these states explicitly. From a deterministic I-diamond automaton $\mathcal{A}$ and a distributed alphabet with process set $\mathcal{P}$, we construct a deterministic asynchronous automaton of size $4^{|\mathcal{P}|^4} \cdot |\mathcal{A}|^{|\mathcal{P}|^2}$. We believe that this complexity, although exponential in the number of processes, is interesting in practice: an implementation of such a device needs only memory of size logarithmic in $|\mathcal{A}|$ and polynomial in $|\mathcal{P}|$. We also show that computing the next state on-the-fly can be done in time polynomial in both $|\mathcal{A}|$ and $|\mathcal{P}|$.

Related work. Besides general constructions of Zielonka type, there are a couple of different constructions, however they either apply to subclasses of regular trace languages, or they produce non deterministic automata (or both). The first category includes [10,3], that provide deterministic asynchronous cellular automata from a given trace homomorphism in case that the dependence alphabet is acyclic and chordal, respectively. These constructions are quite simple and only polynomial in the size of the monoid (thus still exponential in the size of a DFA). In the second category we find [16], who gives an inductive construction for non deterministic, deadlock-free asynchronous cellular automata. (A deadlock-free variant of Zielonka's construction was proposed in [14]). The paper [1] proposes a construction of asynchronous automata of size exponential only in the number of processes (and polynomial in $|\mathcal{A}|$) as our construction, but

it yields non deterministic asynchronous automata (inappropriate for monitoring or control). Notice that while asynchronous automata can be determinized, there are cases where the blow-up is doubly exponential in the number of processes [8].

2 Preliminaries

We fix a finite set $\mathcal{P}$ of processes and a finite alphabet Σ. Each letter $a \in \Sigma$ is an action associated with the set of processes $\mathrm{dom}(a) \subseteq \mathcal{P}$ involved in its execution. A pair (Σ, dom) is called *distributed alphabet*. A deterministic automaton over the alphabet Σ is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, q^0, F\rangle$ with a finite set of states Q, a set of final states F, an initial state q^0 and a transition function $\Delta : Q \times \Sigma \to Q$. As usual we extend Δ to words in Σ^*. The automaton accepts $w \in \Sigma^*$ if $\Delta(q^0, w) \in F$. We use $\mathcal{L}(\mathcal{A})$ to denote the language accepted by $\mathcal{A}$. The size $|\mathcal{A}|$ of $\mathcal{A}$ is the number of its states.

Concurrent executions of systems with shared actions given by a distributed alphabet (Σ, dom), are readily modeled by Mazurkiewicz traces [9]. The idea is that the distribution of the alphabet defines an independence relation among actions $I \subseteq \Sigma \times \Sigma$, by setting $(a, b) \in I$ if and only if $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset$. We call (Σ, I) an *independence alphabet*. The independence relation induces a congruence $\sim$ on Σ^* by setting $u \sim v$ if there exist words $u_1, \ldots, u_n \in \Sigma^*$ with $u_1 = u$, $u_n = v$ and such that for every $i < n$ we have $u_i = xaby$, $u_{i+1} = xbay$ for some $x, y \in \Sigma^*$ and $(a, b) \in I$. An $\sim$-equivalence class is simply called a *(Mazurkiewicz) trace*. We denote by $[u]$ the trace associated with the word $u \in \Sigma^*$ (for simplicity we do not refer to I, neither in $\sim$ nor in $[u]$, as the independence alphabet is fixed). Trace prefixes and trace factors are defined as usual, with $[p]$ a trace prefix (trace factor, resp.) of $[u]$ if p is a word prefix (word factor, resp.) of some $v \sim u$. As usual, we write $\leq$ for the prefix order. For two prefixes T_1, T_2 of T, we let $T_1 \cup T_2$ denote the smallest prefix T' of T such that $T_i \leq T'$ for $i = 1, 2$.

For several purposes it is convenient to represent traces by (labeled) pomsets. Formally, a trace $T = [a_1 \cdots a_n]$ ($a_i \in \Sigma$ for all i) corresponds to a labeled pomset $\langle E, \lambda, \leq\rangle$ defined as follows: $E = \{e_1, \ldots, e_n\}$ is a set of events (or nodes), one for each position in T. Event e_i is labeled by $\lambda(e_i) = a_i$, for each i. The relation $\leq$ is the least partial order on E with $e_i \leq e_j$ whenever $(a_i, a_j) \notin I$ and $i \leq j$. In Figure 1 we give an example for the pomset of a trace T, depicted by its Hasse diagram. The labeling of a total order on E that is compatible with $\leq$ is called a *linearization* of T.

An automaton $\mathcal{A}$ is called *I-diamond* if for all $(a, b) \in I$, and s a state of $\mathcal{A}$: $\Delta(s, ab) = \Delta(s, ba)$. Note that the I-diamond property implies that the language of $\mathcal{A}$ is *I-closed*: that is, $u \in \mathcal{L}(\mathcal{A})$ if and only if $v \in \mathcal{L}(\mathcal{A})$ for every $u \sim v$. This permits us to write $\Delta(s, T)$ where T is a trace, to denote the state reached by $\mathcal{A}$ from s on some linearization of T. Languages of I-diamond automata are called *regular trace languages*.

Definition 1. *A* deterministic asynchronous automaton *over the distributed alphabet (Σ, dom) is a tuple $\mathcal{B} = \langle (S_p)_{p\in\mathcal{P}}, (\delta_a)_{a\in\Sigma}, s^0, Acc\rangle$ where:*

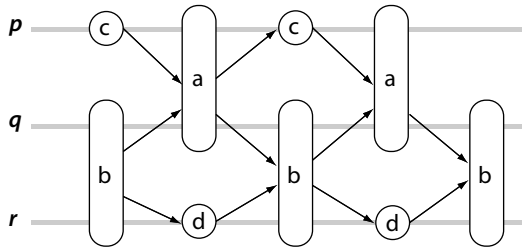

Fig. 1. The pomset associated with the trace $T = [c\,b\,a\,d\,c\,b\,a\,d\,b]$, with $\mathrm{dom}(a) = \{p, q\}$, $\mathrm{dom}(b) = \{q, r\}$, $\mathrm{dom}(c) = \{p\}$, $\mathrm{dom}(d) = \{r\}$.

- *S_p is the finite set of local states of a process $p \in \mathcal{P}$,*
- *$\delta_a : \prod_{p \in dom(a)} S_p \to \prod_{p \in dom(a)} S_p$ is the local transition function associated with an action $a \in \Sigma$,*
- *$s^0 \in \prod_{p \in \mathcal{P}} S_p$ is the global initial state,*
- *$Acc \subseteq \prod_{p \in \mathcal{P}} S_p$ is a set of global accepting states.*

We call $\prod_{p \in \mathcal{P}} S_p$ the set of *global states* (whereas S_p is the set of p-local states). In this paper the size of an asynchronous automaton $\mathcal{B}$ is the total number of *local states* $\sum_{p \in \mathcal{P}} |S_p|$. This definition is very conservative, as one may want to count also Acc or the transition functions (that can be exponential in $|\mathcal{B}|$). We will see that our construction allows to compute both Acc and the transition functions in polynomial time.

With the asynchronous automaton $\mathcal{B}$ one can associate a *global automaton* $\mathcal{A}_\mathcal{B} = \langle Q, \Sigma, \Delta, q^0, Acc \rangle$ where:

- The set of states is the set of global states $Q = \prod_{p \in \mathcal{P}} S_p$ of $\mathcal{B}$, the initial and the accepting states are as in $\mathcal{B}$.
- The transition function $\Delta : Q \times \Sigma \to Q$ is defined by $\Delta((s_p)_{p \in \mathcal{P}}, a) = (s'_p)_{p \in \mathcal{P}}$ with $(s'_p)_{p \in \mathrm{dom}(a)} = \delta_a((s_p)_{p \in \mathrm{dom}(a)})$ and $s'_p = s_p$, for every $p \notin \mathrm{dom}(a)$.

Clearly $\mathcal{A}_\mathcal{B}$ is a finite deterministic automaton with the I-diamond property.

Definition 2. *The language of an asynchronous automaton $\mathcal{B}$ is the language of the associated global automaton $\mathcal{A}_\mathcal{B}$.*

We conclude this section by introducing some basic notations on traces. For a trace T, we denote by $\mathrm{dom}(T) = \bigcup_{e \in E} \mathrm{dom}(\lambda(e))$ the set of processes occurring in T. For a process $p \in \mathcal{P}$, we denote by $\mathrm{pref}_p(T)$ the minimal trace prefix of T containing all events of T on process p. Hence, $\mathrm{pref}_p(T)$ has a unique maximal event that is the last (most recent) event of T on process p. This maximal event is denoted as $\mathrm{last}_p(T)$. Intuitively, $\mathrm{pref}_p(T)$ corresponds to the history of process p after executing T. We extend this notation to a set of processes $P \subseteq \mathcal{P}$ and denote by $\mathrm{pref}_P(T)$ the minimal trace prefix containing all events of T on processes from P. By $\mathrm{last}(T)$ we denote the set of events $\{\mathrm{last}_p(T) \mid p \in \mathcal{P}\}$. For example, in Figure 1 we have $\mathrm{pref}_p(T) = [cbadcba]$ and $\mathrm{last}_p(T)$ is the second a of the pomset. The set $\mathrm{last}(T)$ contains the second a and the third b.

3 Zielonka-type constructions: state of the art

All general constructions of deterministic asynchronous automata basically follow the main ideas of the original construction of Zielonka [15]. These constructions start with a regular, I-closed word language, that is given either by a homomorphism to a finite monoid, or by an I-diamond automaton. In most applications we are interested in the second case, where we start with a (possibly non deterministic) automaton. The general constructions yield either asynchronous automata as defined in the previous section, or asynchronous *cellular* automata, that correspond to a concurrent-read-owner-write model.

Theorem 1. *[15] Let $\mathcal{A}$ be an I-diamond automaton over the independence alphabet (Σ, I). A deterministic asynchronous automaton $\mathcal{B}$ can be effectively constructed with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

We now review the constructions of [2,12,6] and recall their complexities. It is well known that determinization of word automata requires an exponential blow-up, hence the complexity of going from a non deterministic I-diamond automaton $\mathcal{A}$ to a deterministic asynchronous automaton is at least exponential in $|\mathcal{A}|$. In that case, [6] gives an optimal construction as it is simply exponential in both parameters $|\mathcal{A}|$ and $|\mathcal{P}|$. Since determinization has little to do with concurrency, we assume from now on that $\mathcal{A}$ is a deterministic automaton.

- [2] introduces asynchronous mappings and constructs asynchronous cellular automata of size $|\Sigma|^{|\Sigma|^2} \cdot |\mathcal{A}|^{2^{|\Sigma|}}$.
- [12] constructs asynchronous automata of size $|\mathcal{P}|^{|\mathcal{P}|^2} \cdot |\mathcal{A}|^{|\mathcal{A}| \cdot 2^{|\mathcal{P}|}}$.
- [6] introduces zone decompositions and constructs asynchronous automata of size $2^{3|\mathcal{P}|^3} \cdot |\mathcal{A}|^{|\mathcal{A}| \cdot |\mathcal{P}|^2}$.

Comparing our present construction with previous ones, we obtain asynchronous automata of size $4^{|\mathcal{P}|^4} \cdot |\mathcal{A}|^{|\mathcal{P}|^2}$. In all these constructions, the obtained automata are such that every process knows the state reached by $\mathcal{A}$ on its history. We abstract this property below, and show in the following section that our construction is optimal in this case.

Definition 3. *A deterministic asynchronous automaton $\mathcal{B}$ is called* locally rejecting *if for every process p, there is a set of states $R_p \subseteq S_p$ such that for every trace T:*

$pref_p(T) \notin pref(\mathcal{L}(\mathcal{B}))$ iff the p-local state reached by $\mathcal{B}$ on T is in R_p.

Notice that R_p is a trap: if $\mathcal{B}$ reaches R_p on trace T, then so it does on every extension T' of T. Obviously, no reachable accepting global state of $\mathcal{B}$ has a component in R_p. For these reasons we call states of R_p rejecting.

Our interest in locally rejecting automata is motivated by observing that all general constructions [15,2,3,13,12,6] of deterministic asynchronous automata produce such automata. Suppose that $\mathcal{A}$ is a (possibly non deterministic) I-diamond automaton, and $\mathcal{B}$ a deterministic asynchronous automaton produced

by one of the constructions in [15,13,12,6] (a similar statement applies to the asynchronous cellular automata in [2,3]). Then the local p-state s_p reached by $\mathcal{B}$ after processing the trace T determines the set of states reached by $\mathcal{A}$ on $\mathrm{pref}_p(T)$, for every process p. Thus, if no state in this set can reach a final state of $\mathcal{A}$, then we put s_p in R_p. This makes $\mathcal{B}$ locally rejecting.

4 An exponential lower bound

In this section we present our lower bound result. We show that transforming an I-diamond deterministic automaton into a locally rejecting asynchronous automaton may induce an exponential blow-up in the number of processes. For this we define a family of languages Path_n, such that the minimal sequential automaton for Path_n has size $\mathcal{O}(n^2)$ but every locally rejecting asynchronous automaton recognizing Path_n is of size at least $2^{n/4}$.

Let $\mathcal{P} = \{1, \ldots, n\}$ be the set of processes. The letters of our alphabet are pairs of processes, two letters are dependent if they have a process in common. Formally, the distributed alphabet is $\Sigma = \binom{\mathcal{P}}{2}$ with $\mathrm{dom}(\{p, q\}) = \{p, q\}$.

The language Path_n is the set of traces $[x_1 \cdots x_k]$ such that every two consecutive letters have a process in common: $x_i \cap x_{i+1} \neq \emptyset$ for $i = 1, \ldots, k-1$. Observe that a deterministic sequential automaton recognizing this language simply needs to remember the last letter it has read. So it has less than $|\mathcal{P}|^2$ states.

Theorem 2. *Every locally rejecting asynchronous automaton recognizing Path_n is of size at least $2^{n/4}$.*

Proof. Take a locally rejecting automaton recognizing Path_n. Without loss of generality we suppose that $n = 4k$. To get a contradiction we suppose that process n of this automaton has less than 2^k (local) states.

We define for every integer $0 \leq m < k$ two traces: $a_m = \{4m, 4m+1\}\{4m+1, 4m+2\}\{4m+2, 4m+4\}$ and $b_m = \{4m, 4m+1\}\{4m, 4m+3\}\{4m+3, 4m+4\}$. To get some intuition, the reader may depict traces a_0 and b_0 and see that both a_0 and b_0 form a path from process 0 to process 4, the difference is that trace a_0 goes through process 2 while trace b_0 goes through process 3.

Consider the language L defined by the regular expression $(a_0 + b_0)(a_1 + b_1) \cdots (a_{k-1} + b_{k-1})$. Clearly, language L is included in Path_n and contains $2^k = 2^{n/4}$ different traces. As we have assumed that process n has less than 2^k states, there are two different traces t_1, t_2 from L such that process n is in the same state after t_1 and t_2. For simplicity of presentation we assume that t_1 and t_2 differ on the first factor: t_1 starts with a_0, and t_2 with b_0.

We can remark that processes 0 and n are in the same state after reading $t_1\{0, 3\}$ and t_2. For process 0 it is clear as in both cases it sees the same trace $\{0, 1\}\{0, 3\}$. By our hypothesis, process n is in the same local state after traces t_1 and t_2, therefore also after traces $t_1\{0, 3\}$ and t_2.

Consider now the state s_n reached by n after reading $t_2\{0, n\}$. Since $t_2\{0, n\} \in \mathrm{Path}_n$, the state s_n is not in R_n. By the above, the same state s_n is also reached

after reading $t_1\{0,3\}\{0,n\}$. Trace t_1 starts with $a_0 = \{0,1\}\{1,2\}\{2,4\}$ and continues with processes whose numbers are greater than 4, so $\{0,3\}$ commutes with all letters of t_1 except $\{0,1\}$. Hence $t_1\{0,3\} \notin \text{pref}(\text{Path}_n)$. Since trace t_1 ends with an action of process n, we have $\text{pref}_n(t_1\{0,3\}\{0,n\}) = t_1\{0,3\}\{0,n\} \notin \text{pref}(\text{Path}_n)$. Since we have assumed that the automaton is locally rejecting, we should have $s_n \in R_n$. A contradiction. □

5 A matching upper bound

Our goal is to modify the construction from [6] in order to make it polynomial with respect to the size of the sequential automaton. We give an overview of the new construction, first describing the objects the asynchronous automaton manipulates. Some details of the mechanics of the automaton will follow. Overall, although described in a different way, the present construction follows closely [6]. The main difference is the *state information* computed on the zone decomposition of a trace. This state information becomes polynomial (instead of exponential), but its update is much more involved than in [6].

We fix a set of processes $\mathcal{P}$ and a distributed alphabet (Σ, dom). Let $\mathcal{A} = \langle Q, \Sigma, \Delta, q^0, F\rangle$ be a deterministic I-diamond automaton. A candidate for an equivalent asynchronous automaton $\mathcal{B} = \langle (S_p)_{p\in\mathcal{P}}, (\delta_a)_{a\in\Sigma}, s^0, Acc\rangle$ has a set of states for each process and a local transition function. The goal is to make $\mathcal{B}$ calculate the state reached by $\mathcal{A}$ after reading a linearization of a trace T. Let us examine how $\mathcal{B}$ can accomplish this task. After reading a trace T the local state of a component p of $\mathcal{B}$ depends only on $\text{pref}_p(T)$. Hence, $\mathcal{B}$ can try to calculate the state reached by $\mathcal{A}$ after reading (some linearization of) $\text{pref}_p(T)$. When a next action, say a, is executed, processes in $\text{dom}(a)$ can see each others' states and make the changes accordingly. Intuitively, this means that these processes can now compose their information in order to calculate the state reached by $\mathcal{A}$ on $\text{pref}_{\text{dom}(a)}(T)\, a$. To do so they will need some information about the structure of the trace.

As usual, the tricky part of this process is to reconstruct the common view of $\text{pref}_{\text{dom}(a)}(T)$ from separate views of each process: $\text{pref}_p(T)$ for $p \in \text{dom}(a)$. For the sake of example suppose that $\text{dom}(a) = \{p,q,r\}$, and we know the states s_p, s_q and s_r, reached by $\mathcal{A}$ after reading $\text{pref}_p(T)$, $\text{pref}_q(T)$ and $\text{pref}_r(T)$, resp. We would like to know the state of $\mathcal{A}$ after reading $\text{pref}_{\{p,q,r\}}(T)$. This is possible if we can compute the contributions of $\text{pref}_q(T) \setminus \text{pref}_p(T)$ and $\text{pref}_r(T) \setminus \text{pref}_{\{p,q\}}(T)$. The automaton $\mathcal{B}$ should be able to do this by looking at s_p, s_q, and s_r, only. This remark points out the challenge of the construction: find the type information that allows to deduce the behaviour of $\mathcal{A}$, and that at the same time is updatable by an asynchronous automaton.

5.1 General structure

Before introducing formal definitions it may be worth to say what is the general structure of the states of the automaton $\mathcal{B}$. Every local state will be a triple $(ts, ZO, \overline{\Delta})$, where

– ts will be a time stamping information as in all general constructions of asynchronous automata;
– ZO will be a zone order, a bounded size partial order on a partition of the trace;
– $\overline{\Delta}$ will be state information, recording the behavior of $\mathcal{A}$ on the partition given by ZO.

Roughly, we will use time stamping to compute zone orders, and zone orders to compute state information. The latter provides all the necessary information about the behaviour of $\mathcal{A}$ on (a linearization of) the trace.

Time stamping: The goal of the *time stamping function* [15] is to determine for a set of processes P and a process q the set $\text{last}(\text{pref}_P(T)) \cap \text{last}(\text{pref}_q(T))$. This set uniquely determines the intersection of $\text{pref}_P(T)$ and $\text{pref}_q(T)$ (for details see e.g. [13]). Computing such intersections is essential when composing information about $\text{pref}_p(T)$ for every $p \in \text{dom}(a)$ into information about $\text{pref}_{\text{dom}(a)}(T)$. The main point is that there exists a deterministic asynchronous automaton that can accomplish this task:

Theorem 3. *[13] There exists a deterministic asynchronous automaton $\mathcal{A}_{TS} = \langle (S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, s^0\rangle$ such that for every trace T and state $s = \Delta(s^0, T)$ reached by $\mathcal{A}_{TS}$ after reading T:*

> *for every $P \subseteq \mathcal{P}$ and $q, r, r' \in \mathcal{P}$, the set of local states $\{s_p \mid p \in P \cup \{q\}\}$ allows to determine if $\text{last}_r(\text{pref}_P(T)) = \text{last}_{r'}(\text{pref}_q(T))$.*

Moreover, such an automaton can be effectively computed and its local states can be described using $\mathcal{O}(|\mathcal{P}|^2 \log(|\mathcal{P}|))$ bits.

For instance, if a new b is executed after $T = [cbadcbad]$ in Figure 2, process r and processes p, q can determine that the intersection of their last-sets consists of the second b. Indeed, $\text{last}(\text{pref}_{p,q}(T))$ is made of the second a (for $\text{last}_p = \text{last}_q$) and the second b (for last_r). Also, $\text{last}(\text{pref}_r(T))$ is made of the second d (for last_r), the second b (for last_q) and the first a (for last_p).

Zone orders: Recall that one of our objectives is to calculate, for every $p \in \mathcal{P}$, the state reached by $\mathcal{A}$ on $\text{pref}_p(T)$. As the discussion on page 7 pointed out, for this we may need to recover the transition function of $\mathcal{A}$ associated with $\text{pref}_q(T) \setminus \text{pref}_P(T)$ for a process q and a set of processes P. Hence we need to store information about the behaviour of $\mathcal{A}$ on some relevant factors of T that are not prefixes. Zones are such relevant factors. They are defined in such a way that there is a bound on the number of zones in a trace. The other crucial property of zones is that for every extension T' of T and every $P \subseteq \mathcal{P}, q \in \mathcal{P}$, if a zone of T intersects $\text{pref}_q(T') \setminus \text{pref}_P(T')$ then it is entirely in this set. A zone order is an abstract representation of the decomposition of a trace into zones.

Definition 4. *[6] Let $T = \langle E, \leq, \lambda\rangle$ be a trace. For an event $e \in E$ we define the set of events $L(e) = \{f \in \text{last}(T) \mid e \leq f\}$. We say that two events e, e' are equivalent (denoted as $e \equiv e'$) if $L(e) = L(e')$. The equivalence classes of $\equiv$ are called* zones. *We denote by dom(Z) the set of processes active in a zone Z.*

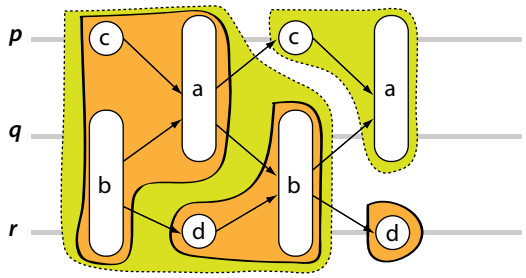

Fig. 2. The three zones of $\mathrm{pref}_r(T)$ (darker) are marked with solid lines. The two zones of $\mathrm{pref}_{\{p,q\}}(T)$ (lighter) are represented by dotted lines.

There is a useful partial order on zones that we define now. Let Z, Z' be two zones of some trace T. We write $Z \lessdot Z'$ if $Z \neq Z'$ and $e < e'$ for some events $e \in Z$, $e' \in Z'$. It is easy to see that $Z \lessdot Z'$ implies that $L(Z') \subsetneq L(Z)$. Thanks to this property we can define the *order on zones*, denoted $Z \leq Z'$, as the smallest partial order containing the $\lessdot$ relation.

Lemma 1. *A trace is partitioned in at most $|\mathcal{P}|^2$ zones.*

The lemma above gives a slightly better bound than [6]. Moreover, it can be shown that its bound is asymptotically optimal. Figure 2 depicts the trace $T = [cbadcbad]$. Recall that $\mathrm{last}(\mathrm{pref}_r(T))$ consists of the first a, the second b and the second d. There are three zones in $\mathrm{pref}_r(T)$: Z_1 contains the first a, b and c, Z_2 the first d and the second b, and Z_3 the second d. We have $Z_1 < Z_2 < Z_3$.

Definition 5. *A* zone order *is a labeled partial order $ZO = \langle V, \leq, \xi \colon V \to 2^{\mathcal{P}} \rangle$, where every element is labeled by a set of processes. We require that every two elements whose labels have non empty intersection are comparable: $\xi(v) \cap \xi(v') \neq \emptyset \;\Rightarrow\; (v \leq v' \vee v' \leq v)$. We say that such a zone order is the* zone order of a trace *T, if there is a bijection μ from V to zones of T preserving the order and satisfying $\xi(v) = dom(\mu(v))$.*

Lemma 2. *The zone order of a trace can be stored in $|\mathcal{P}|^2(|\mathcal{P}|^2 + |\mathcal{P}|)$ space. So there are at most $2^{\mathcal{O}(|\mathcal{P}|^4)}$ zone orders.*

State information: We describe now the state information for each zone of the trace. Let $ZO = \langle V, \leq, \xi : V \to 2^{\mathcal{P}} \rangle$ be the zone order of some trace T, via a bijection μ. For an element $v \in V$ we denote by T_v the factor of T consisting of zones up to $\mu(v)$: that is, the factor covering $\mu(v')$ for all $v' \leq v$. Observe that T_v is a prefix of T. For instance, in Figure 2 the zone order of $\mathrm{pref}_r(T)$ contains three vertices $v_1 < v_2 < v_3$, and T_{v_2} is the trace $[bcadb]$.

Definition 6. *We say that a function $\overline{\Delta} : V \to Q$ is* state information *for the zone order ZO of a trace T if for every v we have $\overline{\Delta}(v) = \Delta(q^0, T_v)$, namely the state of $\mathcal{A}$ reached on a linearization of T_v.*

Observe that a zone order for a trace of the form $\mathrm{pref}_p(T)$ has one maximal element v_p: it corresponds to the last action of p. If $\overline{\Delta}$ is the state information for T, then the state reached by $\mathcal{A}$ on reading a linearization of $\mathrm{pref}_p(T)$ is $\overline{\Delta}(v_p)$.

5.2 The construction of the asynchronous automaton

Let us come back to the description of the asynchronous automaton $\mathcal{B}$. For every $p \in \mathcal{P}$, a local state in S_p will have the form $(ts_p, ZO_p, \overline{\Delta}_p)$. The automaton will be defined in such a way that after reading a trace T the state s_p reached at the component p will satisfy:

- ts_p is the time stamping information;
- ZO_p is the zone order of $\mathrm{pref}_p(T)$;
- $\overline{\Delta}_p$ is the state information for ZO_p.

By [13] we know that $\mathcal{B}$ can update the ts_p component. The proposition below says that $\mathcal{B}$ can update the ZO_p and $\overline{\Delta}_p$ components.

Proposition 1. *Let T be a trace and $a \in \Sigma$ an action. Suppose that for every $p \in dom(a)$ we have the time stamping information ts_p and the zone order with state information $(ZO_p, \overline{\Delta}_p)$ of $pref_p(T)$. We can then calculate the zone order and the state information of $pref_p(Ta)$, for every $p \in dom(a)$.*

We also need to define the sets of rejecting states R_p and the global accepting states Acc of $\mathcal{B}$. Observe that by Proposition 1, from the local state s_p we can calculate $\Delta(q^0, \mathrm{pref}_p(T))$, namely the state of $\mathcal{A}$ reached after reading a linearization of $\mathrm{pref}_p(T)$. This state is exactly the state associated to the unique maximal element of the zone order in s_p. Hence, $\mathcal{B}$ can be made locally rejecting by letting $s_p \in R_p$ if $\Delta(q^0, \mathrm{pref}_p(T))$ is not productive in $\mathcal{A}$, i.e., no final state can be reached from it.

To define accepting tuples of states of $\mathcal{B}$ we use the following proposition:

Proposition 2. *Let T be a trace. Given for every $p \in \mathcal{P}$ the time stamping ts_p, and the zone order ZO_p with state information $\overline{\Delta}_p$ of $pref_p(T)$, we can calculate $\Delta(q^0, T)$, the state reached by $\mathcal{A}$ on a linearization of T.*

In the light of Proposition 2, a tuple of states of $\mathcal{B}$ is accepting if the state $\Delta(q^0, T)$ of $\mathcal{A}$ is accepting. The two propositions give us:

Theorem 4. *Let $\mathcal{A}$ be a deterministic I-diamond automaton over the distributed alphabet (Σ, dom). We can construct an equivalent deterministic, locally rejecting asynchronous automaton $\mathcal{B}$ with at most $4^{|\mathcal{P}|^4} \cdot |\mathcal{A}|^{|\mathcal{P}|^2}$ states.*

We now describe informally the main ingredients of the proof of Proposition 1 (Proposition 2 is proved along similar lines). The zone order ZO of $\mathrm{pref}_{P \cup \{q\}}(T)$ is built in two steps from ZO_P and ZO_q: first we construct a so-called pre-zone order ZO' by adding to ZO_P the zones from $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$ [6]. Then we quotient ZO' in order to obtain ZO. The quotient operation amounts to

merge zones. The difficulty compared to [6] is posed by the update of the state information. Since the state information for the pre-zone ZO' is inconsistent due to the merge, the crucial step is to compute this information on downward closed sets of zones:

Lemma 3. *Let $ZO = \langle V, \leq, \xi \rangle$, $\overline{\Delta}$ be the zone order and state information for a trace T (via the bijection μ). For every downward closed $B \subseteq V$ we can compute the state reached by $\mathcal{A}$ on a linearization of $T_B = \bigcup\{T_v \mid v \in B\}$, using only ZO and $\overline{\Delta}$.*

The proof of the lemma above is based on a nice observation about I-diamond automata $\mathcal{A}$, taken from [2]. It says that for every three traces T_0, T_1, T_2 with $\mathrm{dom}(T_1) \cap \mathrm{dom}(T_2) = \emptyset$, the state reached by $\mathcal{A}$ on a linearization of $T_0T_1T_2$ can be computed from $\mathrm{dom}(T_1)$ and the states reached on (linearizations of) T_0, T_0T_1, T_0T_2, respectively.

We now sketch the proof of the lemma. We first choose some linearization $v_1, \ldots, v_n$ of B. For each i, k with $i \leq k$, let $B_{i,k} = \{v_1, \ldots, v_i\} \cup \{v_j \mid j > i, v_j \leq v_k\}$. For instance, if there are four zones v_1, v_2, v_3, v_4 with $v_1 < v_2 < v_4$, $v_1 < v_3 < v_4$, and $\xi(v_2) \cap \xi(v_3) = \emptyset$, then $B_{1,2} = \{v_1, v_2\}$, $B_{1,3} = \{v_1, v_3\}$, and $B_{2,3} = \{v_1, v_2, v_3\}$.

We show now how to compute inductively $\Delta(q^0, T_{B_{i,k}})$. Notice that the base case is trivial, as $B_{0,k} = \overline{\Delta}(v_k)$ for all k. Let $i \leq n$. Suppose that for all $k \geq i-1$, we know $\Delta(q^0, T_{B_{i-1,k}})$. In particular, note that the states q^{i-1}, q^i reached on $\mu(v_1 \cdots v_{i-1})$ and $\mu(v_1 \cdots v_i)$, respectively, are known (cases $k = i-1$ and $k = i$). We compute now $\Delta(q^0, T_{B_{i,k}})$, for all $k > i$. Two cases arise. If $v_i \not< v_k$ then we apply the observation of [2] to $q^{i-1}, q^i, \Delta(q^0, T_{B_{i-1,k}}), \xi(v_i)$, which yields $\Delta(q^0, T_{B_{i,k}})$. If $v_i < v_k$, then $B_{i-1,k} = B_{i,k}$ and the state $\Delta(q^0, T_{B_{i,k}})$ is already known. At the end of this polynomial time procedure, we have computed $\Delta(q^0, T_B) = \Delta(q^0, T_{B_{n,n}})$.

Remark 1. The automaton $\mathcal{B}$ of Theorem 4 can be constructed on-the-fly, i.e., given the action $a \in \Sigma$ and the local states s_p of $\mathcal{B}$, $p \in \mathrm{dom}(a)$, one can compute the successor states $\delta_a((s_p)_{p \in \mathrm{dom}(a)})$. The question is now how much time we need for this computation. The update of the time stamping and the update of zone orders take time polynomial in $|\mathcal{P}|$. The update of state information can be done in time polynomial in $|\mathcal{P}|$ and linear in the number of transitions of $|\mathcal{A}|$. So overall, we can compute transitions on-the-fly in polynomial time. Similarly, we can decide whether a global state is accepting in polynomial time.

6 Conclusion

In this paper we have presented an improved construction of asynchronous automata. Starting from a zone construction of [6], we have shown how to keep just one state per zone instead of a transition table. This allows to obtain the first construction that is polynomial in the size of the sequential automaton and exponential only in the number of processes.

It is tempting to conjecture that our construction is optimal. Unfortunately, it is very difficult to provide lower bounds on sizes of asynchronous automata. We have given a matching lower bound for the subclass of locally rejecting automata. It is worth to recall that all general constructions in the literature produce automata of this kind. Moreover the concept of locally rejecting automaton is interesting on its own from the point of view of applications.

We conjecture that the translation from deterministic word automata to asynchronous automata must be exponential in the number of processes (where the size means the total number of local states).

References

1. N. Baudru. Distributed Asynchronous Automata. In *CONCUR'09,* LNCS 5710, pages 115–130. Springer, 2009.
2. R. Cori, Y. Métivier, and W. Zielonka. Asynchronous mappings and asynchronous cellular automata. *Information and Computation*, 106:159–202, 1993.
3. V. Diekert and A. Muscholl. Construction of asynchronous automata. In Diekert and Rozenberg [4], pages 249–267.
4. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
5. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.
6. B. Genest and A. Muscholl. Constructing Exponential-Size Deterministic Zielonka Automata. In *ICALP'06,* LNCS 4052, pages 565–576. Springer, 2006.
7. J. G. Henriksen, M. Mukund, K. N. Kumar, M. Sohoni, and P. S. Thiagarajan. A Theory of Regular MSC Languages. *Inf. Comput.*, 202(1):1–38, 2005.
8. N. Klarlund, M. Mukund, and M. Sohoni. Determinizing Asynchronous Automata. In *ICALP'94,* LNCS 820, pages 130–141. Springer, 1994.
9. A. Mazurkiewicz. Concurrent Program Schemes and their Interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
10. Y. Métivier. An algorithm for computing asynchronous automata in the case of acyclic non-commutation graph. In *ICALP'87,* LNCS 267, pages 226–236. Springer, 1987.
11. M. Mukund, K. N. Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *CONCUR'00,* LNCS 1877, pages 521–535. Springer, 2000.
12. M. Mukund and M. Sohoni. Gossiping, Asynchronous Automata and Zielonka's Theorem. Report TCS-94-2, School of Mathematics, SPIC Science Foundation, Madras, India, 1994.
13. M. Mukund and M. A. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. *Distributed Computing*, 10(3):137–148, 1997.
14. A. Stefanescu. *Automatic synthesis of distributed transition systems.* PhD thesis, Universität Stuttgart, 2006.
15. W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.
16. W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In *Symposium on Logical Foundations of Computer Science, Logic at Botik '89, Pereslavl-Zalessky (USSR),* LNCS 363, pages 278–289. Springer, 1989.